FUNCTIONAL REQUIREMENTS FOR DESIGN OF THE SPACE ULTRARELIABLE MODULAR COMPUTER (SUMC) SYSTEM SIMULATOR: INTERIM REPORT

By

R.T. Curran

W.A. Hornfeck

Prepared for

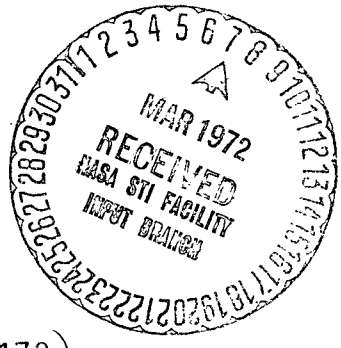
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

George C. Marshall Space Flight Center

Marshall Space Flight Center, Alabama

CONTRACT NAS8-26698

CR-123574



February 1972

CSC

COMPUTER SCIENCES CORPORATION

46R

TECHNICAL REPORT STANDARD TITLE PAGE

1. REPORT NO.	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO. N72-20179
4. TITLE AND SUBTITLE Functional Requirements for Design of the Space Ultrareliable Modular Computer (SUMC) System Simulator: Interim Report		5. REPORT DATE 6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S) R. T. Curran and W. A. Hornfeck		8. PERFORMING ORGANIZATION REPORT #
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Sciences Corporation Field Services Division, Aerospace Systems Center 8300 South Whitesburg Drive Huntsville, Alabama 35802		10. WORK UNIT NO. 11. CONTRACT OR GRANT NO. NAS8-26698
12. SPONSORING AGENCY NAME AND ADDRESS National Aeronautics and Space Administration Washington D. C. 20546		13. TYPE OF REPORT & PERIOD COVERED Contractor Interim Report 14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

Work performed for Marshall Space Flight Center Computation Laboratory.

16. ABSTRACT

This interim report presents the functional requirements for the design of an interpretive simulator for the Space Ultrareliable Modular Computer (SUMC). A review of applicable existing computer simulations is included along with constraints on the SUMC simulator functional design. Input requirements, output requirements, and language requirements for the simulator are discussed in terms of a SUMC configuration which may vary according to the application. Conclusions and recommendations concerning future development are also included.

17. KEY WORDS		18. DISTRIBUTION STATEMENT
Simulation	Digital Computer	Unlimited
Microprogram	Target Computer	
Microinstruction	Host Computer	
Diagnostics	Operating System	B. Hodges, NASA
Subroutine	Link Editor	Contracting Officer

19. SECURITY CLASSIF. (of this report)	20. SECURITY CLASSIF. (of this page)	21. NO. OF PAGES	22. PRICE
Unclassified	Unclassified	45	

MSFC - Form 3292 (May 1969)

FUNCTIONAL REQUIREMENTS FOR DESIGN OF THE SPACE ULTRARELIABLE MODULAR COMPUTER (SUMC) SYSTEM SIMULATOR: INTERIM REPORT

By

R. T. Curran
W. A. Hornfeck

Prepared Under

CONTRACT NAS8-26698

Prepared For

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
GEORGE C. MARSHALL SPACE FLIGHT CENTER
MARSHALL SPACE FLIGHT CENTER, ALABAMA 35812

COMPUTER SCIENCES CORPORATION
8300 South Whitesburg Drive
Huntsville, Alabama 35802

FOREWORD

The studies reported here were administered in the Systems Research Branch, Computer Systems Division, Computation Laboratory, MSFC, with Mr. Bobby C. Hodges as Contracting Officer's Representative. We wish to express our appreciation for Mr. Hodges' help in planning and for his cogent comments and suggestions.

Acknowledgement is also due Mr. James L. Lewis of the Astrionics Laboratory and Mr. Thomas E. Hill of the Computation Laboratory for their assistance in the definition of the simulator functional requirements.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Table

FUNCTIONAL REQUIREMENTS FOR DESIGN OF THE SPACE ULTRARELIABLE MODULAR COMPUTER (SUMC) SYSTEM SIMULATOR: INTERIM REPORT

SUMMARY

This report contains the results of the functional requirements determination phase for an interpretive software computer simulator. This simulator must permit variation of certain design parameters of the target SUMC such as instruction definition, Scratch Pad, main storage and computer word organization and, additionally, must provide the capability to interface with anticipated peripheral device simulators.

The preceding requirements are met by designing software subroutines to accomplish specific SUMC instructions. The subroutines are summoned by a mainline subprogram upon detection of execution commands present in a simulated SUMC program resident in the host computer. Detection is aided by the analysis of the specification parameters for the particular SUMC design.

In addition to the foregoing, the simulator enhances debugging efforts by providing output of the simulated program history such as traces, snapshots and memory dumps. To effect these outputs, the mainline subprogram must detect the appropriate request parameter and cause consequent collection and publishing of the selected program information.

While the SUMC family is the chosen target for simulation, the selection of the host computer must be free of constraints to the feasible limit. For this reason the simulator will be written in the FORTRAN language. In addition to the FORTRAN coding, modularity in the construction of the program will ensure that any computer dependencies will be isolated and easily changed to permit transfer between host computers.

/

SECTION I. INTRODUCTION

This report presents the results of the functional requirements and design definition phase of the study for the generation of the SUMC family simulator. Relevant results of a survey of reports describing active simulators are discussed. A bibliography containing additional references to reports with a general rather than a particular impact on the functional requirements and design of the simulator is included.

Following the review of existing simulators, the simulator environmental constraints are analyzed. Salient simulation-oriented characteristics of the SUMC family target machines are presented. The host machine and simulator operating environment effect on the simulator design is discussed in terms of their effect on the software organization.

Given the definition of the impact of the simulator environment on the simulator functional design, attention is then directed to the simulator itself. The role of the simulator within the SUMC development software is described, and then, input, interpretive simulation and output process functional requirements and design considerations for the simulator are related. Finally, the report is summarized and recommendations are presented.

The myriad problems associated with simultaneous development of the hardware and software for (prototype) computer systems have led to the utilization of large existing computers for software development. Consonant with this trend, software executing on a commercial computer will be provided for the Space Ultrareliable Modular Computer (SUMC) family. An important subset of this software is a computer simulator which permits debugging and verification of SUMC programming in parallel with the computer hardware development.

SECTION II. EXISTING COMPUTER SIMULATORS

A. Review of Existing Simulators

A large number of computer programs presently exist which simulate or emulate* the operation of one computer system on another. Unfortunately, most simulation problems are of a specialized nature which in turn causes simulators to be tailored to individual needs. Many simulation programs make use of special purpose simulation languages, while others employ a standard source language either at the assembly language level or at a much higher level. A further disparity arises from the fact that different levels of simulation may be performed. For example, the simulation of a given target computer may be desired at the machine register level[1]. In this case, all hardware registers and their logical contents are maintained by the host computer throughout the simulation. This type of simulation yields information for the designer which reveals the sequence of target computer hardware operations that take place under a given set of input instructions. On the other hand, the simulation may be at the system level[2]. Here the designer is not concerned with validation of target machine hardware, but with validation of the target computer as an efficient data processing system. For this type of simulation, the designer is interested in CPU loading, I/O channel loading, and application program queues/overruns.

A number of existing simulators have been studied in order to determine their degree of applicability to the problem at hand. While none of the simulators could be used directly in the simulation which is proposed, information and ideas gleaned from these studies should prove valuable in the formulation of a SUMC simulator. The following paragraphs will present brief summaries of the more pertinent papers and reports on existing simulators which have been investigated.

---

*Simulation can be defined as the process of modeling the physical behavior of one machine on another, whereas emulation is defined as a combined hardware-software approach to simulation.

1. Space Ultrareliable Modular Computer (SUMC) Microinstruction Simulator, Tech. Rpt. Space Support Div., Sperry Rand Corp., Huntsville, Alabama.

2. Simulating the Operation of an Aerospace Computer System, E. C. Day, AIAA Paper No. 69-973, September 1969.

1. SUMC Microinstruction Simulator. This simulation program is written in XDS extended FORTRAN IV and uses the Sigma 5 as the host computer. The SUMC is, of course, the target computer and is simulated at the machine register level. This provides a software tool for functionally duplicating SUMC hardware operations specified by the control words in the microprogrammed read-only-memory.

a. The simulator is programmed to:

(1) Perform solution of microinstructions in the same manner as SUMC insofar as possible;

(2) Compute contents of registers, multiplexers, and arithmetic units used to perform microinstruction solutions;

(3) Provide instruction traces or microinstruction traces of computed values for program or microprogram analysis, respectively.

b. The simulator is designed to operate in the following two phases:

(1) Initialization - accepts initial conditions and initiates simulation of application programs;

(2) Simulation - performs application program simulation and outputs information required for analysis.

c. The output information which may be requested by the user includes: Instruction Address Read-Only-Memory (IAROM) dumps, Microprogrammed Read-Only-Memory (MROM) dumps, and Scratch Pad Memory (SPM) dumps. Microinstruction or instruction traces of the elements required for microprogram or program analysis may be output following execution of each microinstruction or instruction. The trace information includes the contents of the Sequence Control Unit, Iteration Counter, Scratch Pad Memory location accessed, Memory Register, Product-Remainder Register, Memory Address Register, Multiplier-Quotient Register, and Exponent Register; the status of the Arithmetic Logic Unit (ALU) overflow, ALU1 carry latch, ALU2 carry latch, and Exponent Arithmetic Logic Unit (EALU) overflow; and also the address of the Scratch Pad Memory location accessed.

2. IBM TC-1 & CP-2 Simulators. These are very similar simulation programs, designed to perform interpretive simulation of flight programs written for the IBM TC-1 and CP-2 aerospace computers. Each simulator provides a means for the dynamic analysis, modification, and control of TC-1 or CP-2 flight programs. In doing so, the simulators assist in debugging operational

modules written for the target computer by precisely duplicating operation of the subject CPU and providing associated I/O device handling.

a. General Design. The simulator acts as a subroutine available to a user control program written in FORTRAN IV with an IBM/360 acting as the host computer. The user control program can communicate information to the simulator and obtain information in return through the following set of interface programs:

(1) User-written simulation control program,
(2) A set of programs comprising the interface programs,
(3) A machine instruction interpreter,
(4) Data collection routines, and
(5) Reporter.

b. The simulated execution of flight programs proceeds in three phases:

(1) Accept initialization parameters (specified by the user control program) and prepare simulator and associated interface programs for execution;

(2) Perform computer simulation and collect information required by programmer (as specified during phase [1];

(3) Process the information collected during phase [2] into a meaningful format.

c. The output information which can be obtained by the user includes:

(1) Dump of requested portion of core in a hexadecimal format;

(2) Output of requested type of trace information; either full trace, block trace or branch trace;

(3) Output of variable quantities whenever a specific location is reached; requires the specification of array of snap addresses, modules in which snap items are found, addresses of snap items desired, and number of snap items;

(4) Comments in the form of a line of EBCDIC information; and

(5) A line of EBCDIC information to be used as a title at the top of each output page.

3. Logicon Generalized Aerospace Computer Simulator. The Logicon Aerospace Computer Simulator (ACS) is an assembly language program which operates on the UNIVAC 1108 under EXEC II. All input to the ACS is handled by the 1108 assembler, which eliminates the need to write elaborate translators or input processors.

a. Basic ACS Simulator. The basic ACS simulator is a skeletal program which becomes complete after the user defines the target computer by assembling its characteristics into the simulator. The four types of input handled by the 1108 assembler are:

(1) Specification of the target computer,
(2) Input of the target computer program,
(3) Construction of the interface routines, and
(4) Specification of the simulator outputs and diagnostics.

b. Operating Modes. The user can select the number and type of operands used by each instruction and the applicability of indexing, extension register usage, and indirect addressing. The target computer may be word or byte oriented with 1's complement, 2's complement, or sign-magnitude arithmetic. Double-precision or floating point arithmetic may be included. Registers may be in memory or may be entirely separate, and any number of registers as well as register types can be included in the target computer specifications.

The simulator may run in an open-loop mode or, through construction of interface routines, in a closed-loop mode with a vehicle/environment simulator. In any event, the target computer program to be executed may be assembled directly using macroinstructions and commands to the assembler in a symbolic format chosen by the user.

c. Output Options. Simulator output options include timing data, error diagnostics, traces, periodic listings of variables, debugging routines, and summaries printed at run termination. A more detailed explanation of the output options available to the user is given in the following paragraphs.

(1) SNAPS - octal printout obtained by keying target computer locations. When such a location-keyed instruction is executed, the contents of the selected target memory location are saved. The items saved are printed for every major or minor loop of the program. User specifies target location to be keyed, target location whose contents are to be saved, the scaling, and the symbolic name of the variable.

(2) Print Function - location-keyed function that prints out the octal contents of one target computer location. User specifies the target computer location to be keyed, the target location to be printed, the scaling, and the symbolic name of the variable. The scaled value is printed along with its symbolic name.

(3) Traces - full trace, in which every instruction executed causes a trace print to occur; partial trace, in which each I/O and/or branch instruction results in a print. Trace outputs may be started or stopped by location-keyed functions, timed inputs, or both. There are 10 trace variables and these can be varied by input. They are:

(a) Host computer instruction address (QHIA)
(b) Operation Code (QOPC)
(c) Operand address 1 (QOPAD1)
(d) Operand 1 (QOPNO1)
(e) Target computer accumulator (QACCUM)
(f) Target computer Q-register (QREQ)
(g) Blank field (QBLNK)
(h) Blank field (QBLNK)
(i) Blank field (QBLNK)
(j) Master clock in cycles (QMTIME)

(4) Data Tapes - SNAP print sets may be written on tape in addition to being printed.

(5) Clocks - six clocks are available for timing execution between selected instructions and are controlled by location-keyed functions. A clock may be turned on or read.

(6) Error Diagnostics - as each input or simulation error is discovered, an error diagnostic is printed indicating the type of error and action taken.

(7) Summary Information - max and min values of snapped variables are printed; the number of times each instruction is used and its simulated op code are printed; a dump of the target memory is printed. An asterisk before the word indicates that an instruction has been executed from that location.

B. Other Related Computer Simulators

The simulation studies described in the previous section merit special attention since they represent interpretive simulators having many of the characteristics envisioned for SUMC. However, some other existing simulators, which are not directly related to the SUMC simulation, have also been investigated. Table I contains information on the most prominent features of these simulators.

C. Summary and Conclusions

The simulator envisioned for use in checkout of SUMC programs will be constructed to handle the simulated machine at a block diagram level. This is

TABLE 1. OTHER RELATED SIMULATORS

SIMULATOR	SIMULATION LANGUAGE	LEVEL OF SIMULATION	TARGET COMPUTER	HOST COMPUTER	INPUT	OUTPUT	COMMENT
CISCO	PL/1-ORIENTED	REGISTER	GENERAL	GENERAL	DESCRIPTION OF EQUIPMENT	TRACE INFORMATION	CISCO COMPILER, LOADER, AND INTERPRETER ARE INCLUDED AS PART OF SIMULATION PACKAGE.
SELMA	MADCAP	INTERPRETIVE	SEL-810A	MANIAC II	SEL-810A MACHINE LANGUAGE PROGRAMS	DIAGNOSTICS	EFFICIENCY HAS NOT BEEN STRESSED.
LICOS	FORTRAN	INTERPRETIVE	LIBRASCOPE GUIDANCE COMPUTER MODELS 1 & 3	IBM 7090	LIBRASCOPE MACHINE INSTRUCTIONS	TRACE INFORMATION, FLIGHT MATRIX	HOST MACHINE HARDWARE IS USED WITH SIMULATION RESULTS WHICH ARE NOT BIT-FOR-BIT.
CSS/360	CSS/360	FUNCTIONAL	IBM SYSTEM/ 4 PI-EP MULTI - PROCESSOR	IBM SYSTEM/ 360	COMPUTATION REQUIREMENTS, EQUIPMENTS BEING MODELED	TIMING INFORMATION, PROGRAM AND I/O QUEUES, LOADING PARAMETERS	SPECIFICALLY TAILORED TOWARD MODELING COMPUTER FUNCTIONS.
SELF-REPAIRING DIGITAL COMPUTER SIMULATOR	IBMAP	BLOCK DIAGRAM	PROTOTYPE SELF-REPAIRING COMPUTER	IBM 7094	MACHINE DESCRIPTION, MEMORY MAP, CONTROL PROGRAM	SNAPSHOTS OF "STATE" OF THE TARGET MACHINE	STANDARD FORTRAN ROUTINES ARE USED FOR I/O FUNCTIONS.

contrasted with a functional simulation which is too far removed from the hardware for the purposes of this project or to a logic simulation which leads to excessively slow simulation and yields results which require considerable reduction. The SUMC simulator will be an interpretive simulator in which all programmable registers are maintained. The simulator will also maintain a memory map of the target computer and simulation results will precisely duplicate those which would be obtained by the target machine.

The SUMC simulator will resemble certain existing simulators which have been developed for aerospace computers. The TC-1 and CP-2 simulators are examples of interpretive simulators which have been developed for use in checkout of flight software. The Logicon Generalized Aerospace Simulator is another example of interpretive simulation designed to provide target computer program checkout. However, the SUMC simulator will be unique in the sense that simulation of a family of SUMC machines will be possible. In addition, the simulator will be programmed in a high-level language so that host machine independence is achieved to the fullest practical extent.

SECTION III. CONSTRAINTS ON THE SIMULATOR FUNCTIONAL DESIGN

A. Baseline Target SUMC

The simulator must have the capability to model the SUMC family. This computer family is expected to consist of computers with design parameters that may vary with the envisioned application. Of particular interest to the simulator design are the SUMC definition parameters given in table 2. The simulator must provide and maintain the contents of pseudo registers and storage locations that appear to be identical to these parameters. These pseudo locations must be updated by the simulator at the level of visibility of the programmer.

1. Storage Organization. The target storage for simulation programs of the SUMC will consist of a main or bulk storage in which programs and data reside with an accompanying scratch pad memory (SPM) which contains various register contents, status indicators, the program counter, and other variables with requirements for rapid access. To ensure generality in the simulation no other assumptions concerning storage will be made. Specific storage organizations will be provided as input parameters to the simulator.

2. Register Organization. Although register organization is, strictly speaking, a subset of storage organization since register contents reside in SPM, conceptually the various classes of registers can be considered logical entities for purposes of discussion of their organization. The simulator must accept descriptive input, discussed more fully later, that specifies this register organization and format for an arbitrary SUMC. A typical pseudo register representation of a SUMC is shown in figure 1.

Conversion of the descriptive input into the pseudo register organization will be discussed later. This conversion must be valid for the range of parameters shown in table 2 to permit the simulator to accept the variety of SUMC designs anticipated for future space-oriented missions.

3. SUMC Instructions. Computer commands defined by the Op Code segment of an instruction are implemented by software subroutines. Each subroutine, in addition to performing SUMC operations at programmer level of visibility, must update the elapsed time counter for the program and must contain any unique error response procedures. The initial version of the simulator will have the capability for executing the basic SUMC commands[3].

---

3. IBM Systems Reference Library, IBM System/360 Principles of Operation, GA22-6821-8, Systems Development Division, Product Publications, Poughkeepsie, N. Y., November 1970.

Table 2. Critical SUMC Design Parameters

	Minimum	Maximum	Increment
Word Length (bits)	16	32	4
Accumulators	1	16	1
Base Registers	1	16	1
Index Registers	1	16	1
General Registers	0	16	1
Scratch Pad Memory (words)	64	TBD*	TBD
Main Storage (words)	128	32,768	TBD

*TBD To be defined

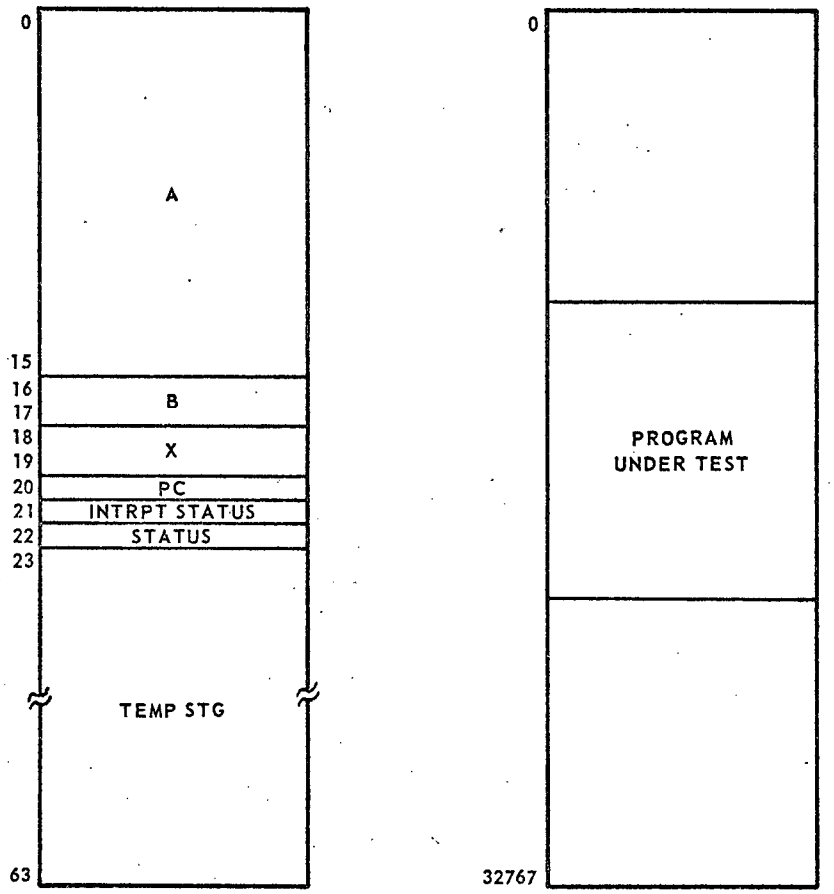


Figure 1. SUMC SCRATCH PAD AND MAIN STORAGE LAYOUT

TYPICAL HOST FORTRAN REPRESENTATION OF SUMC STORAGE

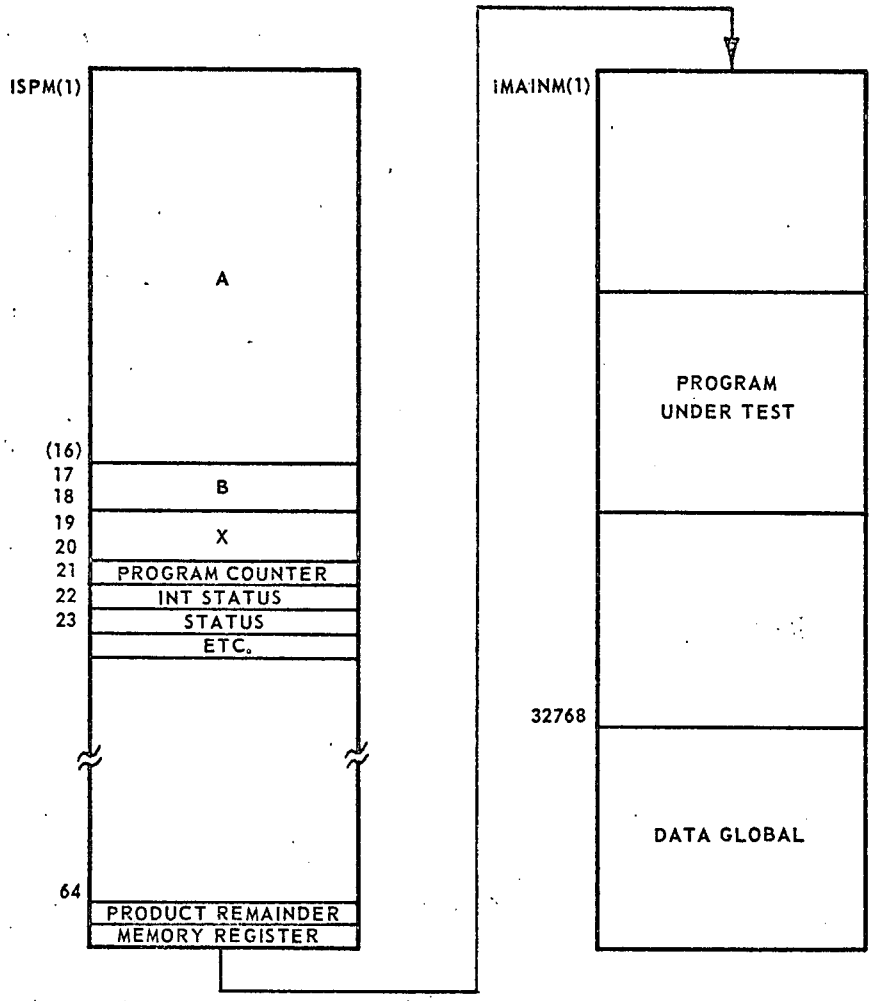


Figure 1. SUMC SCRATCH PAD AND MAIN STORAGE LAYOUT (CONT'D)

4. Input/Output (I/O). The simulator must perform I/O operations in conjunction with a software interface and peripheral device simulation subroutines as shown in figure 2. The simulator will maintain the appropriate status information in the COMMON scratch pad memory representation and will additionally place and retrieve data in the Product Remainder Register (PRR) location in COMMON. Peripheral Device simulation subroutines act in a corresponding, cooperative manner. Peripheral device elapsed time will be maintained by the peripheral device simulation subroutine.

Handshaking protocol must be defined as more specific information becomes available. Presently the I/O process simulator will:

a. Place the datum in the PRR,
b. Set/Reset appropriate status, and
c. Issue a CALL to the specified peripheral device simulator.

Upon completion, control will be relinquished to the simulator.

5. Interrupt Procedures. The simulator will emulate the SUMC sequence of detecting and responding to an interrupt as demonstrated in figure 3. Prior to each instruction FETCH the simulator will interrogate the scratch pad memory (SPM) interrupt status word for presence of interrupts. If interrupts are present, each bit in the word will be examined for an interrupt request (=1). For each of the above interrupt requests a response subroutine must be provided. This response subroutine will be called, and the request bit reset. When all interrupts have been serviced, the simulator is free to FETCH the next in-line program instruction.

This interrupt simulation procedure allows program debugging in order that an operating system with its allied applications programs can be verified using the simulator.

B. Simulator Operating Environment

Presently the simulator is thought of as executing in a batch mode under the operating system of large commercial computers. Candidate computers are the UNIVAC 1108 for production, and the IBM 7094 for development. This difference in host computers implies a design goal of machine independence to the feasible limit. This goal conflicts with the requirement that the simulator utilize host machine hardware to a high degree in the execution of target computer instructions. Candidate host computers must be studied to assess the impact of computer hardware differences on utilization of host computer hardware in a machine-independent environment.

Initial versions of the simulator will run in the batch mode as an applications program and will be self-contained; for instance, I/O statements will be

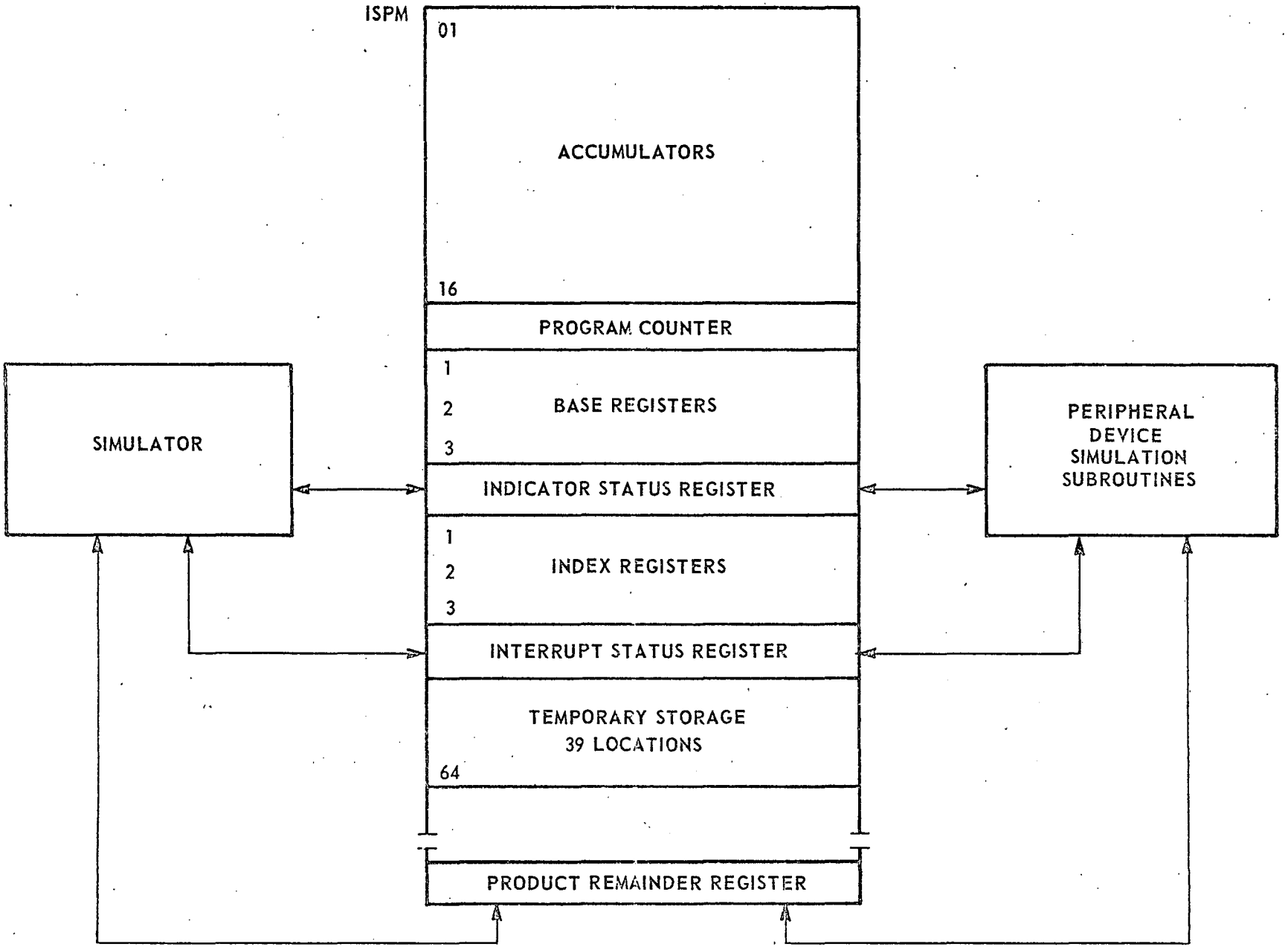


Figure 2. SIMULATOR–PERIPHERAL DEVICE COMMUNICATION

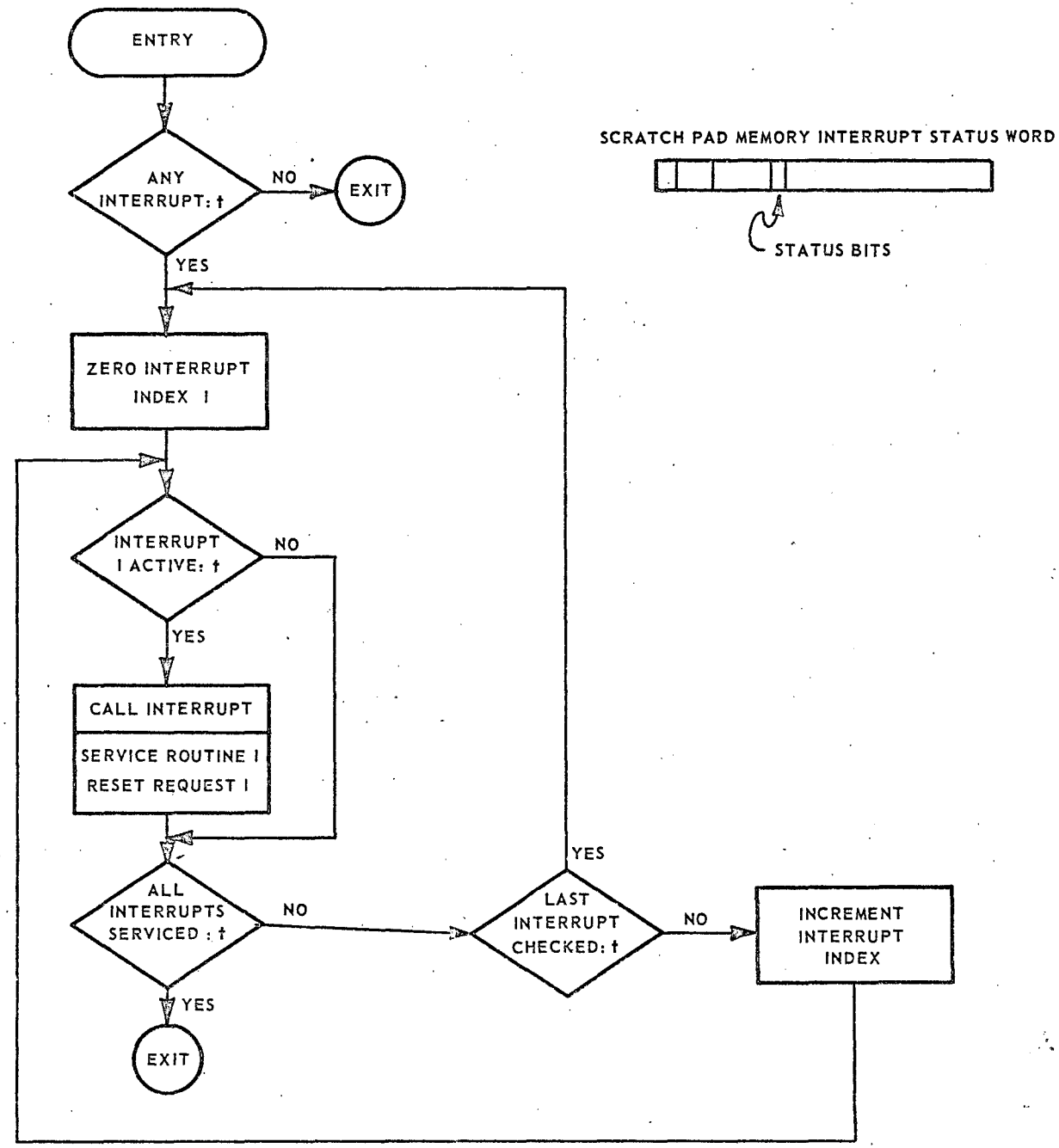


Figure 3. INTERRUPT DETECTION & SERVICE SUBROUTINE

utilized for job description input. Later versions may utilize job control language options of the operating system to perform input of job description parameters such as snap, trace, or dump functions. Early versions of the simulator will exhibit minimum interactions with the host computer operating system, while later versions may utilize facilities provided by the more sophisticated operating systems. Additionally, the requirement for program transfer among several computer installations impacts program modularity. Specifically, the simulator must be constructed as a set of quasi-independent modules, regulated by a controlling module as shown in figure 4 and defined in table 3. Host computer dependence must be constrained to the minimum number of these modules, preferably to one module. This strategy permits transfer of the simulator among host computers with minimum interference and program modification.

C. Simulator Development Environment

Development efforts will take place utilizing the IBM 7094 computer as the testing environment. This model computer, however, is not expected to be the primary host computer for production versions of the simulator. The fact that production host computers will be different from the development host computer impacts the trade study relating machine independence, utilization of host computer hardware, and program execution time as noted. The following will denote specific instances in which carefully detailed functional design is required to circumvent conflicts generated by the chosen development process.

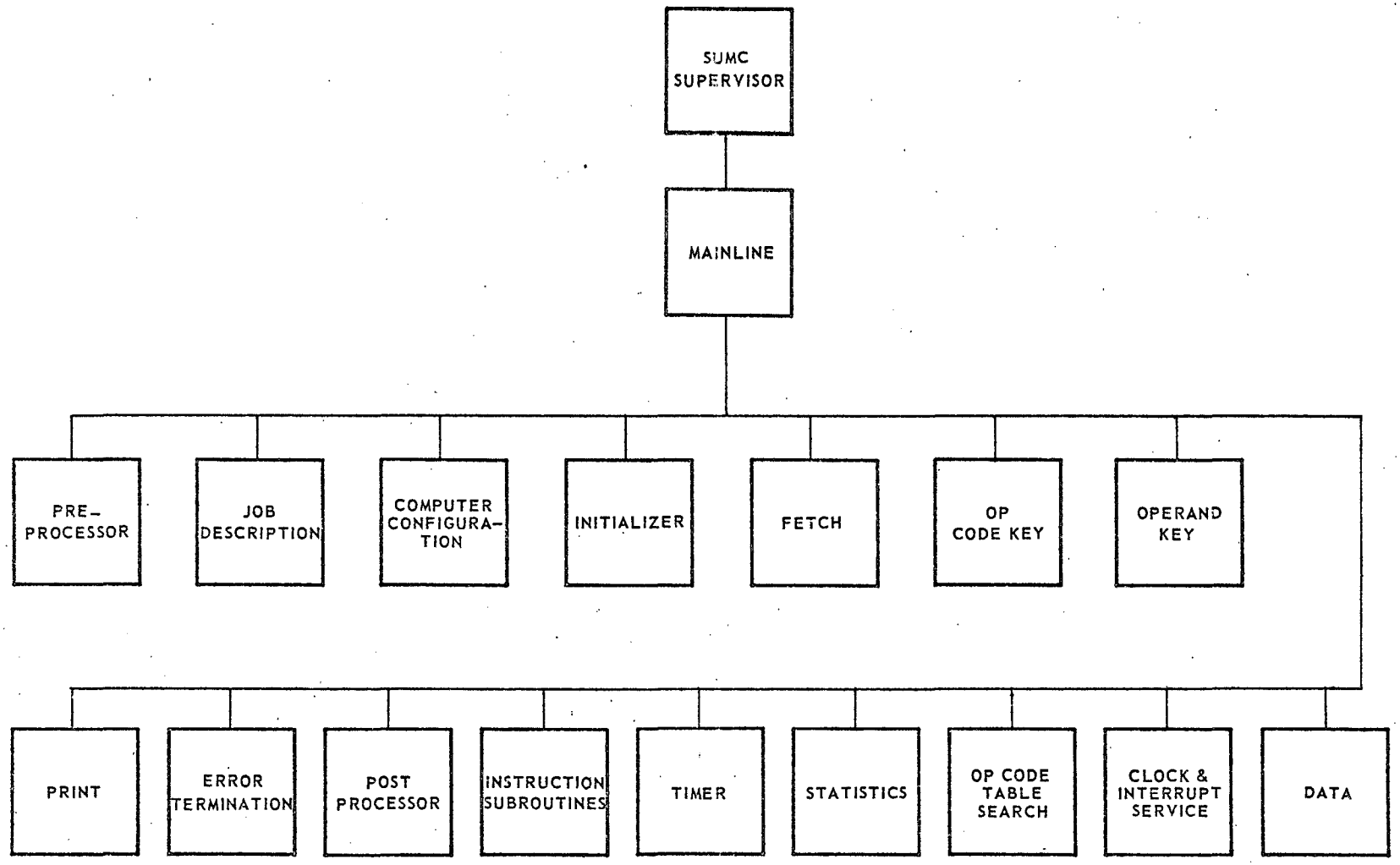
The input development language will be FORTRAN IV[4]. This high level language is a descendant and extension in capability of FORTRAN. This language capability is considered intermediate between FORTRAN II and FORTRAN V. For development of the simulator on the IBM 7094, FORTRAN IV provides the capability of FORTRAN II, with the additional ability to use logical variables and to perform single bit manipulation. Relevant to us, FORTRAN V, available on the UNIVAC 1108, would permit byte (subword) designation and operation, an important enhancement for construction of the simulator.

Considerable effort has been invested in the development of additional supporting software for the IBM 7094[5]. Results of this work were carried forward into the implementation of the support software for the UNIVAC 1108.

---

4. IBM Systems Reference Library, IBM 7090/7094 IBSYS Operating System Version 13, FORTRAN IV Language, GC28-6390-4, IBM Programming & Publications, New York, N. Y., November 1968.

5. MSFC, Marshall Space Flight Center, Preliminary Reference Manual for the IBM 7094, Huntsville, Alabama.



NOTE: TABLE 3. CONTAINS A FUNCTIONAL SUMMARY FOR EACH MODULE

Figure 4. SIMULATOR MODULARITY DIAGRAM

Table 3. Program Module Definition

Program Modules	Function Summary
Preprocessor	Reads output of the Link Editor and transforms it to the appropriate internal host machine format for execution by the simulator.
Job Description	Reads input parameters designating the selection of trace options, run size, execution time limits, storage boundaries and other pertinent information associated with this particular simulation run.
Computer Configuration	Inputs descriptive parameters defining the SUMC configuration to be considered the target machine for this simulation.
Initializer	Sets the initial conditions in simulation variables, e.g. SPM, based on input parameters and Link Editor output, and builds key tables.
Mainline	Controls execution of other program modules including overlay control, if required.
Fetch	Gets the next target computer instruction for execution, extracts sub-fields for Op Code, Registers and Operand definition. Checks interrupt status.
Op Code Key	Searches the Op Code request table for action based on Op Code key, calls appropriate response subroutine.
Operand Key	Similar to Op Code key, above, but searches the Operand request table.
Op Code Table Search	Searches Op Code table for match with instruction Op Code, performs error checks, calls response subroutines.
Instruction Subroutines	Target computer instruction set is simulated at the level of visibility to the programmer.
Timer	Maintains a cumulative record of calculated target machine elapsed time for program execution.
Print	Prints output as requested by a calling subprogram.
Error Termination	Handles abort situations arising from anomalies in the program under simulation.
Post Processor	Handles termination of a simulation run outside of the failure mode. Run parameters are output based on selection. Utilizes Summary routine, Print routine.
Statistics	Tabulates and outputs pertinent program measurement parameters such as number of executions of a specific instruction.
Interrupt Service	Provides the response demanded by an interrupt condition.
Data	Data required by all program modules (global) will be available in COMMON and will include SPM, Main Storage and subroutine communication variables. Temporary storage and data unique to a particular sub-program may be allocated within the sub-program.

We must utilize this IBM 7094 software for enhancement of the basic FORTRAN IV input language and for a similar carry-over to the U1108 to permit a closer approach to machine independence.

SECTION IV. FUNCTIONAL REQUIREMENTS AND DESIGN CONCEPTS

A. Methodology Overview

1. Simulator Interplay with SUMC Development Software. The SUMC interpretive simulator is an important member of a well-conceived set of programs devoted to support of the SUMC family as shown in figure 5, Integration of Simulator into SUMC Support Software. As indicated, the simulator must operate under control of a subexecutive in the UNIVAC 1108 operating system. Programming generation will proceed from translation of input statements, either high level language or assembler statements, into relocatable modules; relocatable modules are put together and converted to a computer load module by the Link Editor. The load module is executed on a SUMC, either a hardware SUMC or a host computer-microcode-software entity simulating a SUMC. The simulator must enact the role of the software portion of the above entity.

2. Simulation Mode. The simulator will operate interpretively by examining each target computer instruction to determine the functional response required and subsequently accomplishing the functional requests through use of the host hardware. Fidelity in yielding the correct result for an arbitrary instruction is the criterion rather than fidelity in executing the precise SUMC sequence to obtain the result. Determination of the validity of a result will occur at the level of visibility to the programmer. This means that, during an execution sequence, the simulator must maintain the contents of computer storage that are available to the programmer and is not responsible for registers, status indicators and other computer information storage not available for reference by the programmer.

3. Simulation Process. The simulator requires a Link Editor or other identically formatted input of the program under test accompanied by further job description and computer configuration identification parameters, which are utilized to select the trace options and to establish the pseudo register and storage layout representing the target SUMC.

A set of program modules required to accomplish the simulation functions and a functional logic flow diagram for the overall interpretive simulation is given in the Appendix. The process of simulation occurs in the following phases:

a. Input of job description, computer configuration, and program memory map;

b. Definition of the target SUMC configuration;

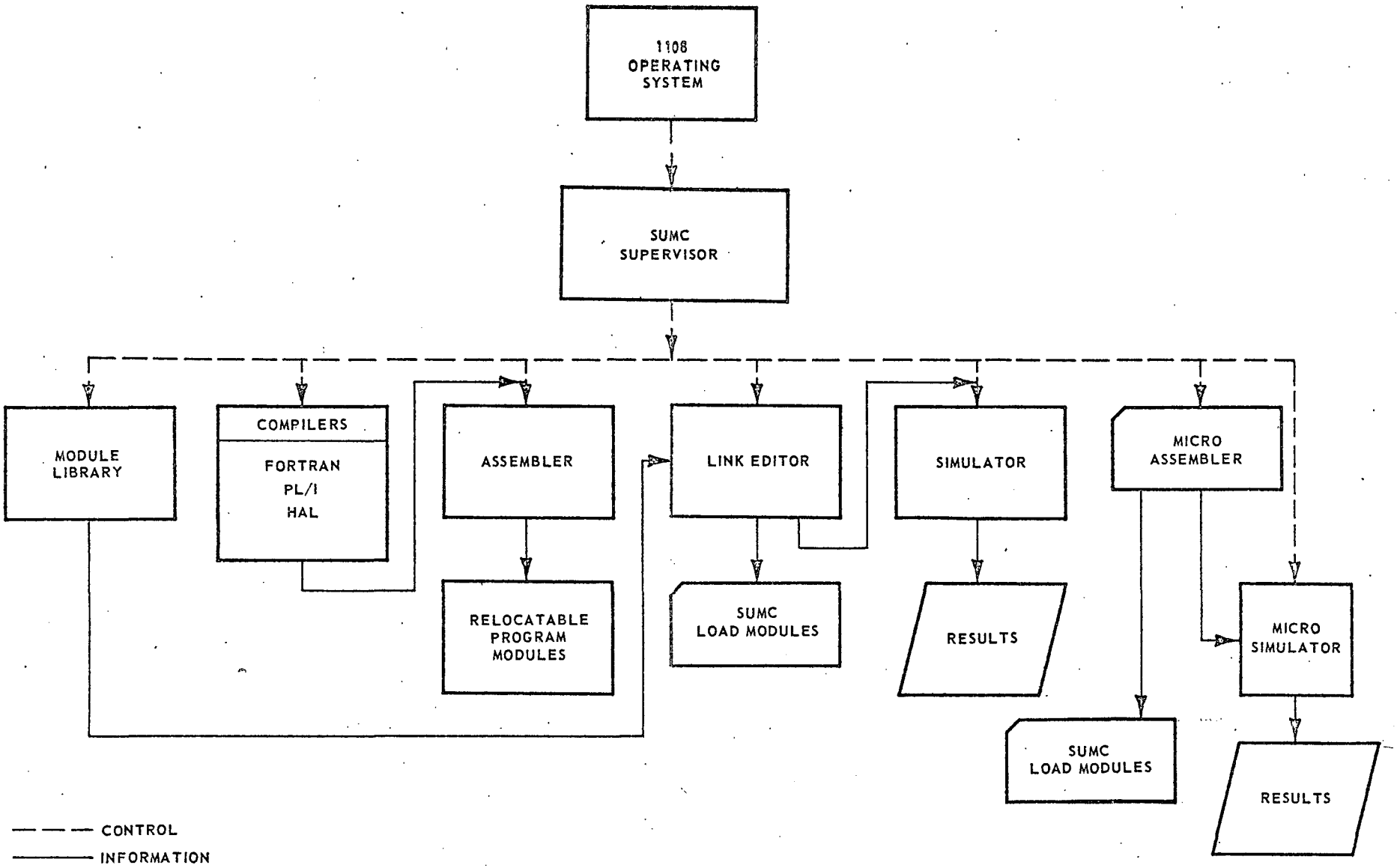


Figure 5. INTEGRATION OF SIMULATOR INTO SUMC SUPPORT SOFTWARE

c. Decomposition of target program instructions, register identification and effective address calculation;

d. Error checking;

e. Trace, Dump and Snap implementation;

f. Instruction interpretive execution;

g. Printout of requested results and/or error notification; and

h. Exit to SUMC Supervisor upon detection of end of program.

The level of effort required for implementation of the above functions is dependent upon the sophistication of the debugging options provided. Simulation execution times are similarly dependent.

B. Input Requirements

1. Computer Configuration. The SUMC has been designed as a highly modular machine and is therefore capable of being configured in a great number of different ways. This characteristic presents a unique problem in the simulation of such a machine in that the simulator must be capable of processing flight programs for any particular SUMC configuration. The simulator for the SUMC family of machines must therefore be designed such that the user, with a minimum of effort, is able to specify the exact SUMC configuration or environment in which the flight programs will be run.

a. Word Length. Since the SUMC is a 4-bit byte modular computer, different configurations may have different word lengths. It will be necessary to design the simulator so that SUMC programs may be run which have been written for 16, 20, 24, 28, or 32-bit word length machines. It will of course be required that the user supply information on target machine word length.

b. Scratch Pad Memory Size and Organization. The organization of the SUMC Scratch Pad Memory (SPM) is another area of design which is inherently flexible. The designer may choose to maintain a minimum of SPM or he may choose to include a rather extensive SPM layout. In addition, the registers which make up the SPM may be organized in many different ways. These SPM registers are programmable registers and as such will be maintained by the SUMC simulator. This will require a user-supplied definition of SPM size, in words, and also SPM register layout.

c. Main Memory Size and Organization. The size of the SUMC main memory will be variable from one configuration to the next. It will therefore be the responsibility of the user to specify target machine main memory

size in some way. This will be the only memory specification needed if it is assumed that main memory consists of a fixed number of contiguous storage locations. However, if it becomes necessary to partition main memory into blocks of contiguous storage, the memory specification problem will become considerably more complex.

d. Floating Point Instructions. The SUMC may be configured to include a floating point arithmetic unit which adds considerable depth to the associated instruction set. Since the absence of a floating point unit results in a much less extensive instruction set, the set of floating point instructions will be included in the target machine instruction set only when user input requirements specify floating point capability.

e. Special Instructions. The simulator will perform the execution of target machine instructions through the use of subroutines. The basic capabilities of the SUMC will be simulated using a fixed set of instruction subroutines. However, additional subroutines may be called by the main program in the event that target machine capabilities dictate their usage. The addition of floating point routines as described in the previous section is an example of this. In a similar manner, any other special instructions such as Input/Output which must be included as part of the simulator must be specified as part of the input requirements. In addition, any special instructions which are specified must be present in the complete library of instructions which are available to the simulator. It should be pointed out here that the SUMC is a microprogrammable machine and therefore could be capable of implementing a virtually infinite number of machine instructions. The possibility would certainly exist that a special instruction is not included in the library of subroutines, in which case an error message could be output to the user.

2. Job Definition. In addition to specifying the SUMC configuration, the simulation job definition must also be supplied to the simulator. In defining a particular simulation, the user must provide:

a. Target program entry point and boundaries,
b. Target machine initial conditions,
c. Simulation output, and
d. Number of simulation runs and time for each.

The user's method for specifying the SUMC program entry point will simply consist of a first word address which is supplied for each target program to be run. The simulator will read the designated first word address, make an appropriate transformation to obtain the host machine address, and begin the simulation with an instruction FETCH from this location.

The user will provide target machine initial conditions for all volatile registers by means of a map of SUMC Scratch Pad Memory. This is possible

since the SUMC architecture provides for location of all programmable registers in Scratch Pad Memory.

The output information provided by the simulator will depend heavily on the specific data which is requested by the programmer. A number of different output types, such as memory dumps and instruction traces, will be available to the user. A number of different options will also be available for each basic type of output. A detailed explanation of the types of output which will be available is given in the next section.

Finally, in the event that a number of simulations are to be run in succession, the user must specify the number of runs and also the maximum allowable run-time for each simulation. A maximum run-time will be designated in terms of both simulated elapsed time and host machine run-time.

3. Link Editor. The SUMC Link Editor will be capable of generating a complete map of the target machine main memory. This simulated main memory map will be the primary input to the SUMC simulator. Once the simulator is provided with the initial conditions for SPM along with the address of the first target instruction, simulated execution of SUMC flight programs will proceed according to the instructions and data present in the SUMC simulated main memory.

C. Output Requirements

1. General. The output specifications for the SUMC simulator will include both standard forms and special forms of diagnostic aids. Since the purpose of the simulator is primarily one of testing and checkout of SUMC flight programs, output diagnostics will perform one of the most important functions of the simulation program. The SUMC simulator will make available to the programmer several different types of output diagnostic aids. Each type of output will also have a number of options available so that different levels of output information can be obtained from the program at the discretion of the user. The following paragraphs will describe the output options which will be available for use when simulating SUMC programs.

2. Dumps. The user will be concerned with memory dumps for both the simulated SUMC main memory and Scratch Pad Memory. Three types of memory dumps will be available to the user, all of which yield a listing of specified memory contents in an octal format. They are:

a. SPM Dump. An output listing of the octal contents of each SPM location. An SPM dump will be available to the user following the completion of any target machine instruction.

b. MM Dump. An output listing of the contents of each main memory location. A main memory dump will be available to the user following

termination of a target machine program regardless of the cause of the termination.

c. Block MM Dump. This output listing will be the same as the complete main memory dump with the exception that only the memory information between selected locations will be dumped.

3. Traces. Several different types of instruction traces will be available for use when the contents of certain key registers are of interest to the programmer.

a. When an instruction trace is in effect, the following information will be outputted after execution of each applicable SUMC instruction:

(1) Instruction Register contents,

(2) Program Counter,

(3) Next Program Counter,

(4) Contents of Accumulator, Base and Index Registers used by the instruction,

(5) Operand Addresses and Operands used by the instruction, and

(6) Contents of Status Registers.

b. The four different types of traces which the user will have the option of using are:

(1) Block Trace. Contents of key registers will be outputted after each instruction execution in a specified block of instructions.

(2) JUMP Trace. Contents of key registers will be outputted after execution of each JUMP instruction.

(3) TEST Trace. Contents of key registers will be outputted after execution of each TEST instruction.

(4) Keyed Trace. Contents of key registers will be outputted after each instruction which involves a keyed op code, address, or address contents.

4. Snaps. This debugging aid is similar to the keyed instruction trace in that information is outputted whenever a specified op code or address is

encountered. The contents of the same key registers will be outputted, but in addition, the contents of programmer specified main memory locations will be printed. The user will provide snap codes and arrays of snap addresses. Addresses of snap items desired and number of snaps desired must also be specified.

5. Error Messages. As each input or simulation error is discovered, an error diagnostic will be printed indicating the type of error and action taken. The error messages will always be outputted in the event of program recognizable errors and the user will not have the option to delete error messages.

D. Language Requirements

The SUMC simulator will be designed so that a number of different target computer configurations may be simulated and also so that a number of different host computers may be employed in performing the simulation. The latter has an impact on simulator source language requirements. In order to meet the requirement of host machine independence, a high level language must be used; however, the source language which is chosen must also be acceptable for processing on many different host machines. FORTRAN IV will be used as the simulator source language since this language comes closest to meeting both of the stated requirements. Although FORTRAN IV is fairly universal in its use, careful attention will still be given to maintaining machine independence since FORTRAN program processing capabilities differ from one machine to the next.

E. Future Simulator Enhancement

The basic simulator is designed to be independent of the host computer operating system. Given a system supervisor program for all SUMC development software, the simulator can be modified to utilize features unique to this supervisor. Allocation of some of the functional responsibilities from the simulator to the supervisor permits amplifying programming capabilities.

One promising area for this shift concerns the problem of causing a "snap" using variable names in place of addresses. If the system supervisor can recognize job control commands such as SNAP, then a possible program input could be the following:

```
SNAP COSBETA, ALPHA, SINGAM.
```

This command could cause output of the current values of the variables COSBETA, ALPHA, and SINGAM to a designated output device. The system supervisor would handle all variable symbolic translations. Similar features would allow ready programmer accumulation of other debugging data using variable names as operands for the system supervisor.

In considering future development paths for the simulator, the use of an on-line time-shared terminal must not be ignored. To utilize the terminal effectively, capability must be added to permit the programmer to interact with the simulator during execution. The programmer must have the power to halt simulation, examine or modify any location, and to perform any function present in the initial version of the simulator.

As the SUMC development proceeds, requirements for high data rate peripherals such as disks, drums and CRT displays can be expected. Such considerations, while not directly affecting the simulator functional design, impact to the extent that the design must permit ready expansion at some future data dependent on SUMC support software development trends.

SECTION V. CONCLUSIONS AND RECOMMENDATIONS

Study of active and historical efforts in simulating computers coupled with experience in similar applications indicates that the most fruitful implementation process is stepwise. The SUMC family interpretive simulator, therefore, will be first implemented as a modular program designed to allow convenient expansion. The initial version will constitute a basis for an extended version which possesses additional capability noted in table 4, Phased Simulator Development.

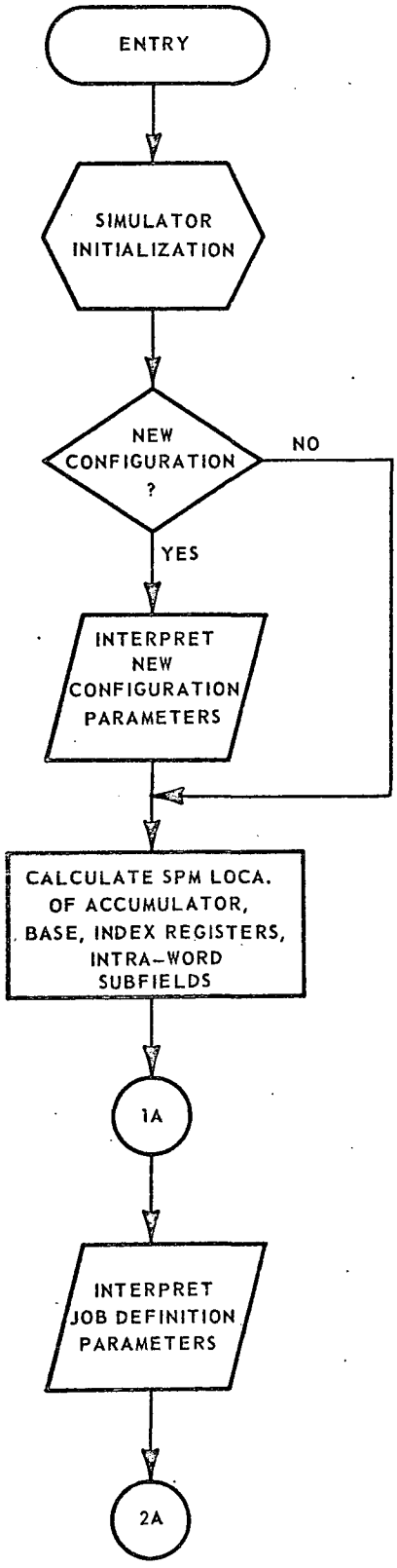
CSC believes that the SUMC interpretive simulator can permit verification of SUMC software with considerably less effort than would be required by utilization of the prototype computer hardware.

Table 4. Phased Simulator Development

	Functional Requirements
Initial Simulator	Simulation of SUMC family for Scratch Pad Memory - 64 words - Main Storage - up to 32K words SUMC word length varying from 16-32 bits Basic instruction repertoire(3) Interrupt capability Input/Output interface capability Snap, Block Trace, Jump Trace and Test Trace Open-ended for convenient expansion Capable of debugging initial systems software
Extended Simulator	Expansion of basic instruction repertoire Assembly language coding of segments of the simulator Detailed design and implementation of peripheral device simulation subroutines Other capabilities noted in previous section Modifications deemed necessary by operational experience

APPENDIX: SUMC SIMULATOR ANNOTATED FLOW CHART

The operation of the Simulator is depicted in the following flowchart (figure A-1). Note that the operations required to execute the interpretive simulation are shown along with explanatory comments to clarify functions of the Simulator.

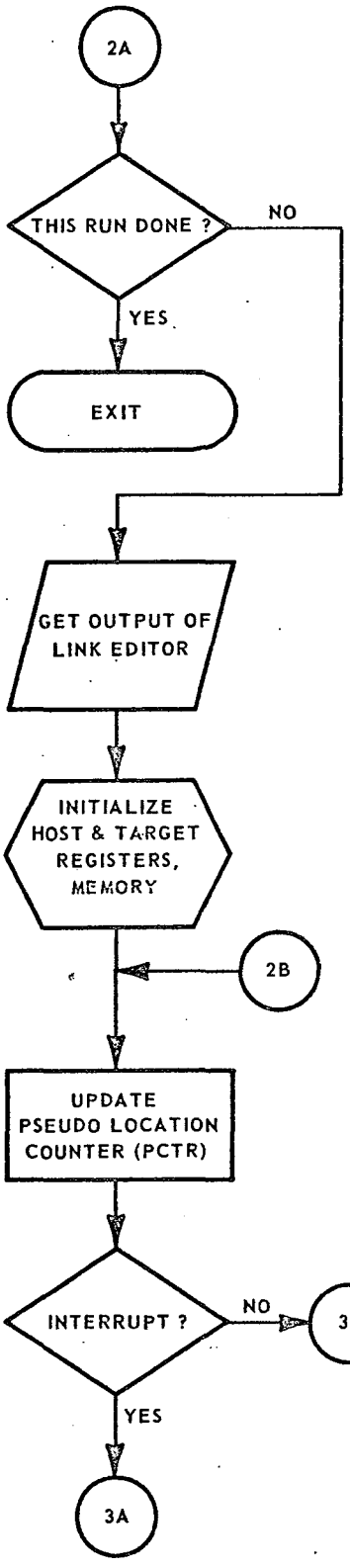


IF NEW PARAMETERS ARE INPUT USING JOB CONTROL LANGUAGE (JCL) CARDS, THEN READING IS UNNECESSARY. IF INPUT IS RESPONSIBILITY OF PROGRAM, THEN CARDS MUST BE READ.

BASED ON INPUT PARAMETERS, PSEUDO REGISTERS (SPM LOCATIONS) ARE MAINTAINED IN HOST MAIN STORAGE.

IF PARAMETERS ARE INPUT VIA JCL, THEN READING IS UNNECESSARY. INPUT PARAMETERS DESCRIBE NUMBER OF RUNS, TRACE, SNAP REQUIREMENTS, RUN TIME LIMITATIONS, AND OTHER DESCRIPTIVE VARIABLES REQUIRED FOR EACH RUN.

FIGURE A-1. SIMULATOR INTERPRETIVE OPERATION (SHEET 1 OF 7)



SIMULATION REQUESTS CAN BE STACKED.

RETURN TO THE SYSTEM SUPERVISOR.

IF COMPUTER CONFIGURATION PARAMETERS ARE TRANSFERRED USING THE LINK EDITOR, THEN PRECEDING INPUT METHOD MUST BE REVISED. LINK EDITOR OUTPUT MAY BE TRANSFORMED TO A MORE CONVENIENT FORMAT BY THE PREPROCESSOR ROUTINES.

ZERO STORAGE AND RELOAD REGISTERS AS DEFINED BY INPUT PARAMETERS.

SUMC UPDATES COUNTER BEFORE FETCHING AN INSTRUCTION.

MUST CHECK INTERRUPT STATUS WORD IN SPM.

FIGURE A-1. SIMULATOR INTERPRETIVE OPERATION (SHEET 2 OF 7)

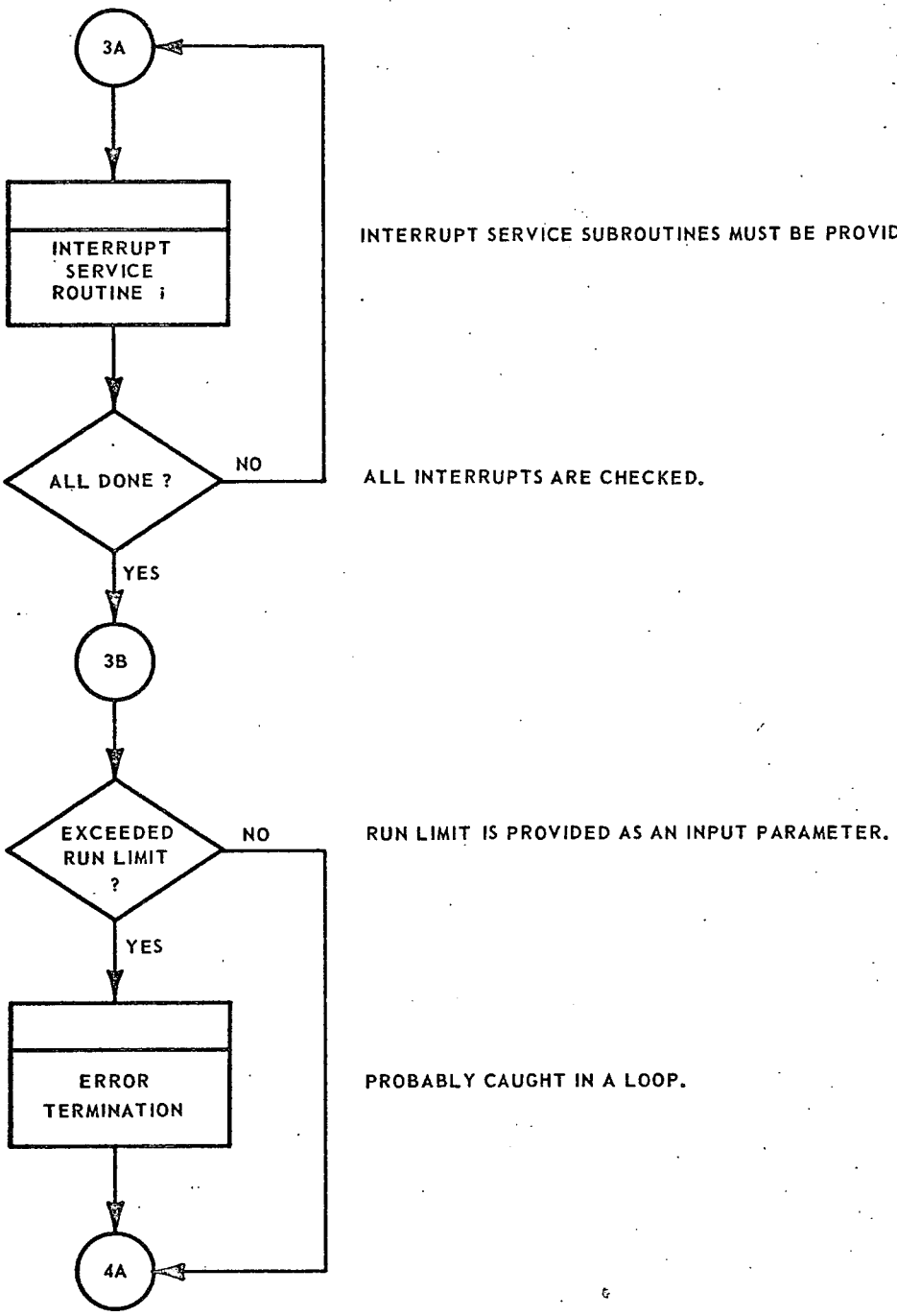


FIGURE A-1. SIMULATOR INTERPRETIVE OPERATION (SHEET 3 OF 7)

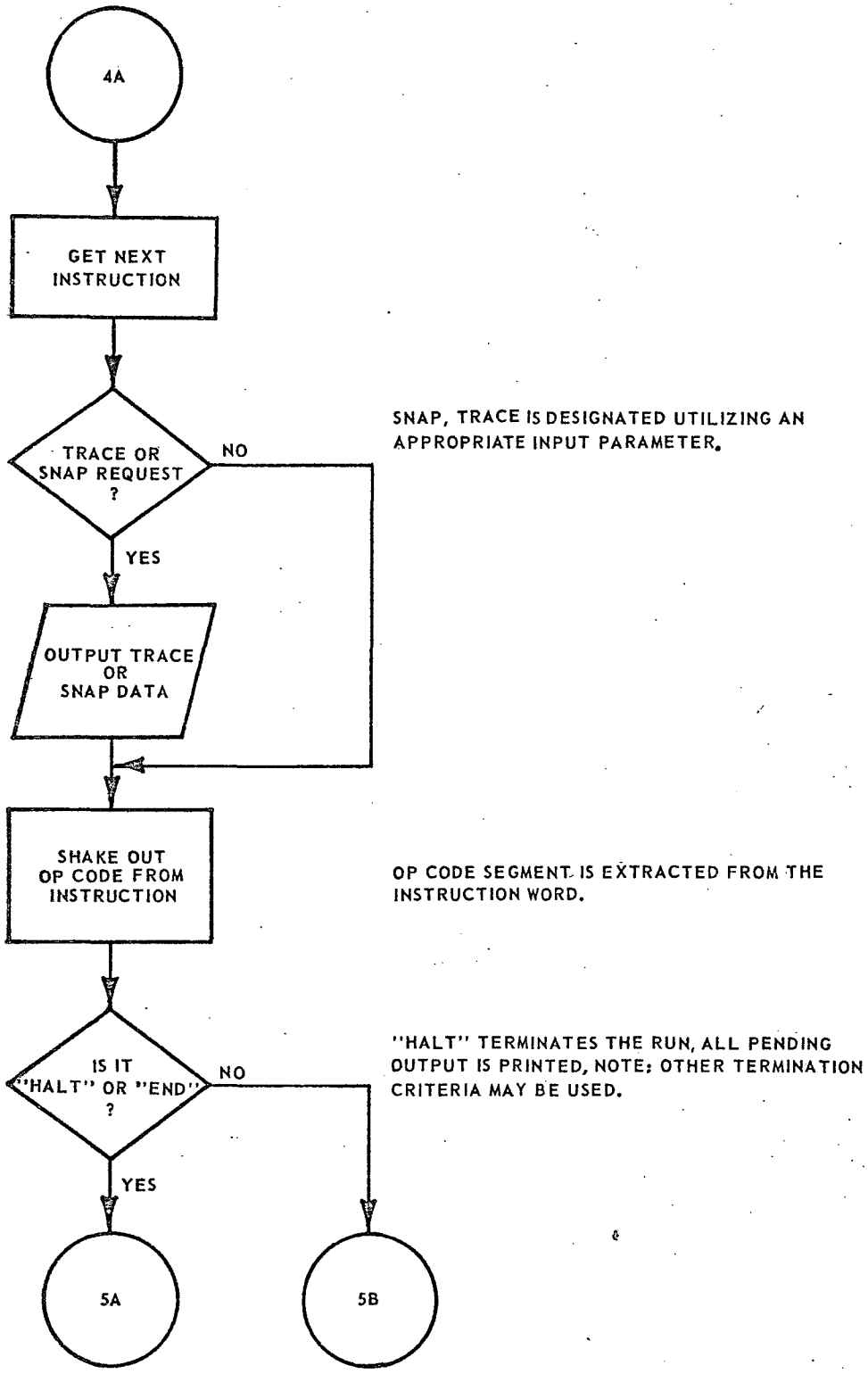


FIGURE A-1. SIMULATOR INTERPRETIVE OPERATION (SHEET 4 OF 7)

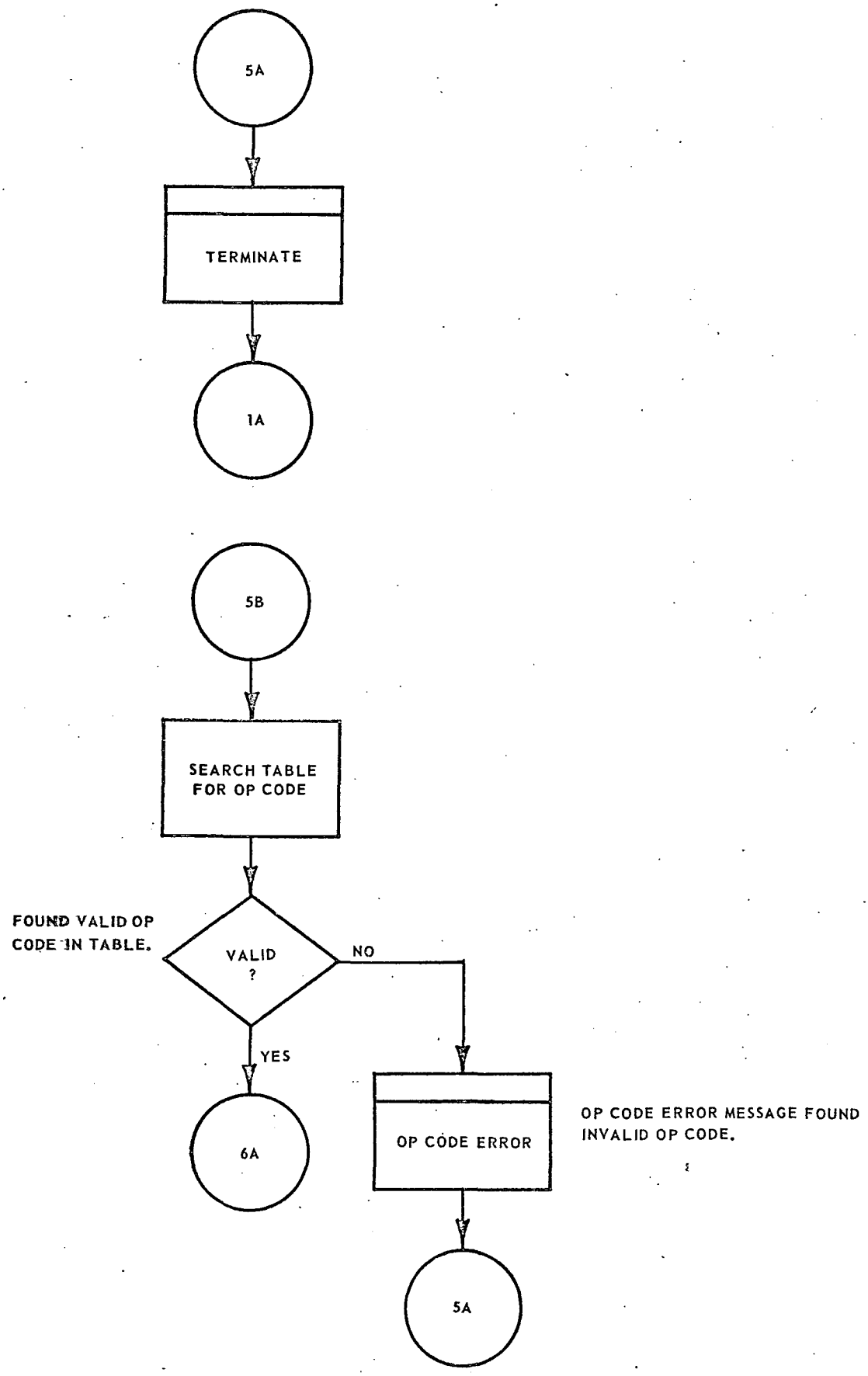
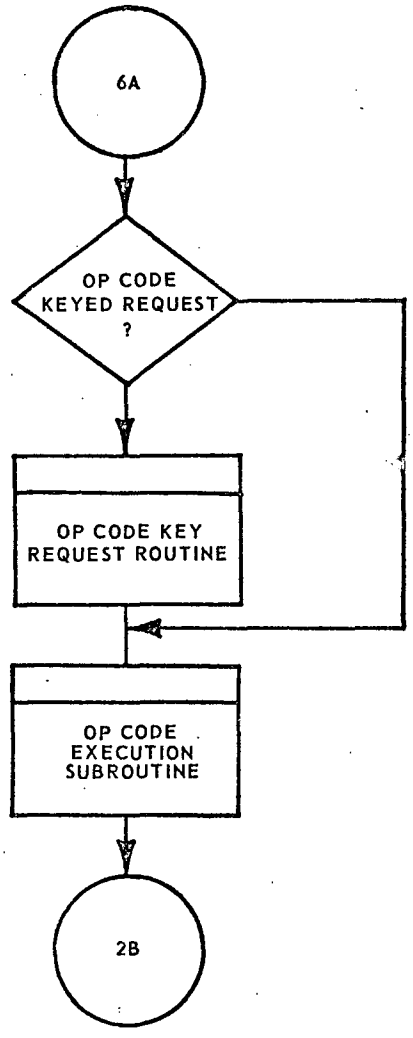
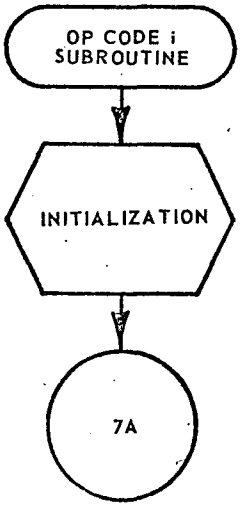


FIGURE A-1. SIMULATOR INTERPRETIVE OPERATION (SHEET 5 OF 7)



* PREDEFINED ACTION FOR PRESENCE OF A SELECTED OP CODE.

EACH NON-BASIC INSTRUCTION WILL BE IMPLEMENTED AS A SUBROUTINE. BASIC INSTRUCTIONS CAN BE CODED IN LINE (SUBROUTINE LINKAGE OVERHEAD GREATER THAN EXECUTION TIME).



A TYPICAL INSTRUCTION WILL BE EXECUTED USING THE FOLLOWING SEQUENCE OF OPERATIONS.

FIGURE A-1. SIMULATOR INTERPRETIVE OPERATION (SHEET 6 OF 7)

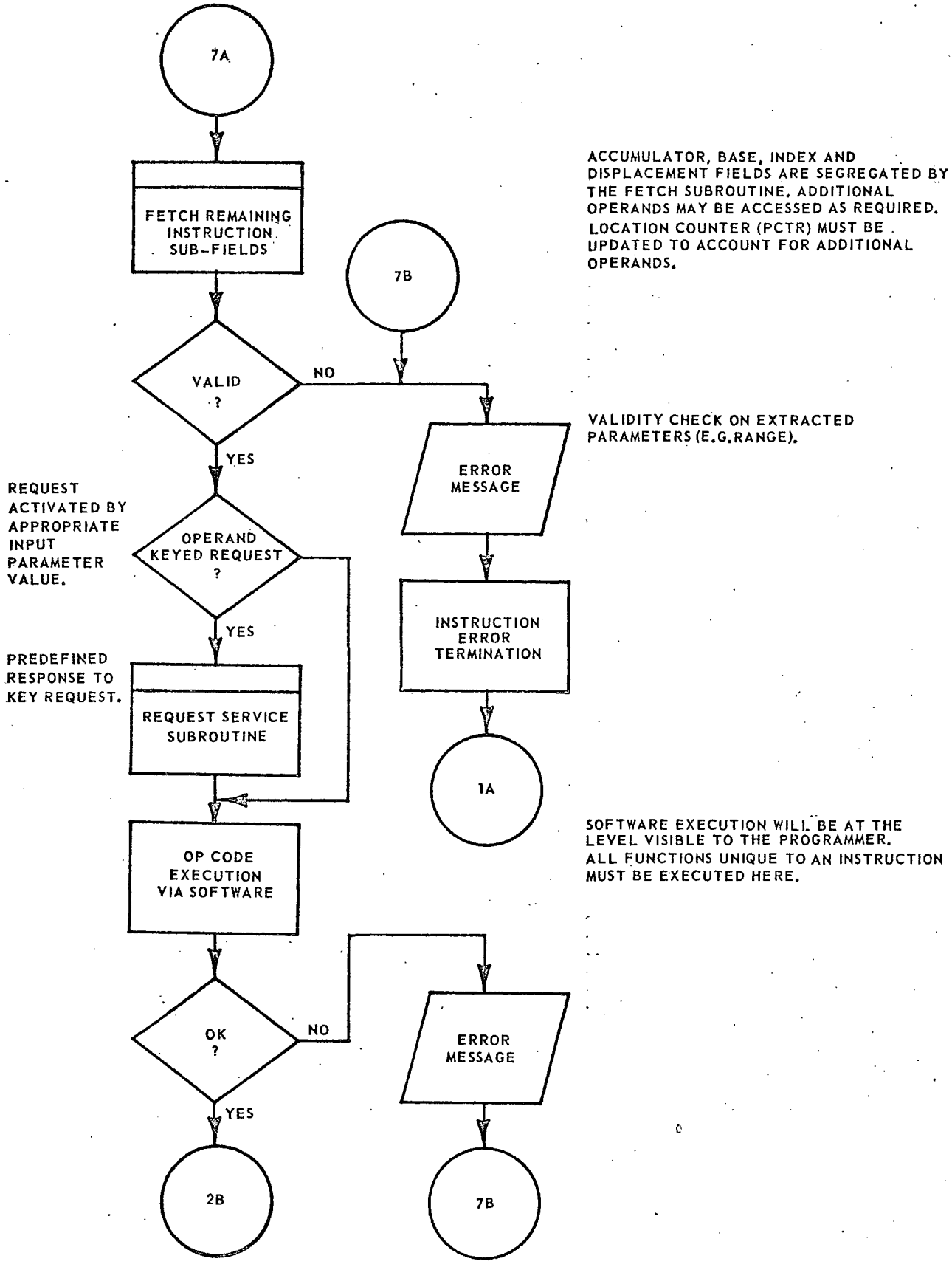


FIGURE A–1. SIMULATOR INTERPRETIVE OPERATION (SHEET 7 OF 7)

REFERENCES

1. Techniques in the Generation of Support Software for the SUMC Functional Requirements, Tech. Rpt., MDAC Contract No. NAS8-27202, August 19, 1971.

2. Techniques in the Generation of Support Software for the SUMC - Preliminary Design Specification, Tech. Rpt., MDAC Contract No. NAS8-27202, August 19, 1971.

3. Program Description - Generalized Aerospace Computer Simulator, Tech. Rpt. No. CS-6921-RO128, Logicon, August 1969.

4. Shuttle Computation System, E. Eastin, Contractor Rpt. SP-233-0252 prepared for MSFC by Sperry Rand Corp. under NASA Contract No. NAS8-20055, Huntsville, Ala., June 8, 1970.

5. MSFC Advanced Aerospace Computer, E. Eastin et al, Contractor Rpt. SP-232-0384 prepared for MSFC by Sperry Rand Corp. under NASA Contract NAS8-20055, Huntsville, Ala., July 6, 1970.

6. Proposed Instruction Set for SUMC System, E. Thompson et al, Contractor Rpt. SP-232-0405-1 prepared for MSFC by Sperry Rand Corp. under NASA Contract NAS8-20055, Huntsville, Ala., September 4, 1970.

7. LICOS - An Interpretive Simulation of the Librascope Computer Model-1 and Model-3, M. McLennan, Contractor Rpt. prepared under NASA Contract NAS3-3232, December 20, 1963.

8. Simulation of the SEL-810A Computer on Maniac II (SELMA), N. Metropolis et al, Report prepared by Los Alamos Scientific Laboratory of the University of California, Los Alamos, New Mexico, November 1966.

9. A Burroughs 220 Emulator for the IBM 360/25, T. A. Schoen and M. R. Belsole, Jr., IEEE Transactions on Computers, July 1971.

10. Microprogramming: Principles and Practices, S. S. Husson, Prentice-Hall, Englewood Cliffs, N. J., 1970.

11. IBM Systems Reference Library, IBM 7090/7094 Support Package for IBM System/360, C28-6501-2, IBM Programming & Publications, New York, N. Y., 1964.

12. FORTRAN IV Computing and Applications, R. L. Nolan, Addison-Wesley Publishing Co., Inc., 1971.

13. Programming the IBM 7090, J. A. Saxon, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1963.

14. Computer Software, I. Flores, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1965.

15. Investigation and Simulation of a Self-Repairing Digital Computer, I. Terris, University Microfilms, Inc., Ann Arbor, Mich., 1965.

16. IBM Systems Journal, Volume Eight, Number Four, 1969, pp. 251-350.

17. The Use of Simulation in the Design of a Real-Time, Multiprocessing Computer System, C. E. Price, Fourth Annual Simulation Symposium Record of Proceedings, Gordon and Breach Science Publishers, New York, N. Y., 1971.

18. The PMS and ISP Descriptive Systems for Computer Structures, C. G. Bell and A. Newell, Proc. AFIPS Spring Joint Computer Conf., 1970, pp. 351-374.

19. Evaluation of Aerospace Computer Architecture, M. D. Anderson and V. J. Marek, Contractor Rpt. prepared under NASA Contract NAS12-589.

20. Modular Design of Computers through the Use of Multi-Level Simulation, K. A. Duke et al, Rpt. RC2872 (#13504), IBM Thomas J. Watson Research Center, Yorktown Heights, N. Y., May 8, 1970.

21. Evaluation of Hardware-Firmware-Software Trade-offs with Mathematical Modeling, H. Barsamian and A. De Cegama, Proc. AFIPS Spring Joint Computer Conf., 1971, pp. 151-162.